# A formal approach for network security policy validation

Fulvio Valenza[1, 2*], Tao Su[1], Serena Spinoso[1], Antonio Lioy[1], Riccardo Sisto[1], and Marco Vallini[1]
[1]*Politecnico di Torino, Dip. di Automatica e Informatica, Torino, Italy*
[2]*CNR-IEIIT, c.so Duca degli Abruzzi 24, Torino I-10129, Italy*
{first.last}@polito.it

**Abstract**

Network security is a crucial aspect for administrators due to increasing network size and number of functions and controls (e.g. firewall, DPI, parental control). Errors in configuring security controls may result in serious security breaches and vulnerabilities (e.g. blocking legitimate traffic or permitting unwanted traffic) that must be absolutely detected and addressed. This work proposes a novel approach for validating network policy enforcement, by checking the network status and configuration, and detection of the possible causes in case of misconfiguration or software attacks. Our contribution exploits formal methods to model and validate the packet processing and forwarding behaviour of security controls, and to validate the trustworthiness of the controls by using remote attestation. A prototype implementation of this approach is proposed to validate different scenarios.

**Keywords**: network security policy, policy conflict analysis, policy validation, remote attestation.

## 1 Introduction

In recent years, the adoption of server virtualization, Network Function Virtualization (NFV), and cloud computing techniques has brought several advantages such as service provisioning and deployment depending on user's requests. This approach enables elastic capacity to add or remove services reducing hardware cost. Although these techniques have several benefits, the management complexity of the entire system increases. During the last ten years, several approaches (in the field of Policy-Based Network Management, PBNM) have been proposed to automatically configure services and applications. This typically provides automatic configuration of applications and services from scratch by defining high-level policies. Although this permits automatic provisioning of resources, by hiding refinement process details, these approaches typically do not support the management of enforced configurations, that is a crucial and complex task in production environments (e.g. data center). It requires high accuracy (e.g. to identify and perform precise modification on configuration settings) and it must limit service downtime. Although the automatic deployment of an application instance is a common feature of recent virtualization platforms, on the other side, the monitoring and management of its configuration is currently not well addressed. A typical provisioning system does not periodically check the configuration settings of a instance. Often, this operation is performed manually, by an administrator, in case of failure or misbehaviour. Think, for example, of updates to a firewall configuration. This approach has at least two drawbacks. First of all, it is an error-prone and expensive task because it is performed by humans. For example, the administrator manually adds a new rule on a firewall that shadows another existing rule, modifying the resulting policy in an unexpected way. Second, it does not address misconfigurations that do not affect service operation. For example, an attacker could add a rule on a firewall to mirror traffic to another system. In this case, the service operates correctly but its configuration is altered and the attack

*Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 8:1 (Mar. 2017), pp. 79-100
*Corresponding author: Politecnico di Torino, Dip. di Automatica e Informatica, Corso Duca degli Abruzzi, 24, 10129 Torino , Tel: +39-(0)11-090-7192

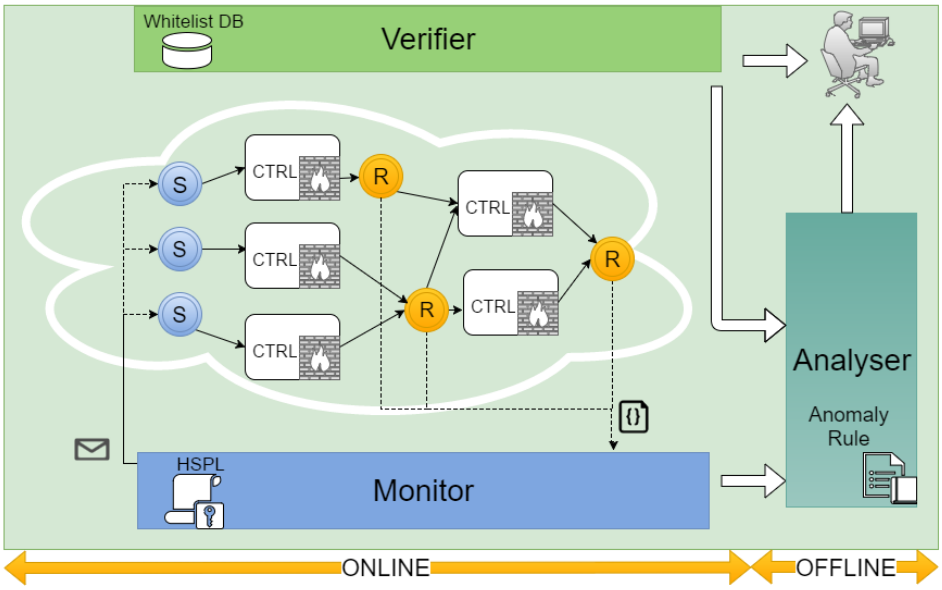



Figure 1: Policy validation workflow and the involved components.

succeeds. Therefore, to detect and avoid these situations, especially for complex scenarios (e.g. data centre), automatic monitoring and analysis of service configurations is mandatory.

Commonly resources are virtualised (e.g. a virtual machine, VM, rather than a physical machine) and run on a specific platform, namely the *hypervisor* on a dedicated node or as part of a cloud computing environment. However, this additional software layer increases the attack surface: for example, an attacker could modify a software component to change its behaviour (e.g. to steal information from a virtual resource). Also in this case, these software changes do not affect the service operation, thus an automatic approach is needed. Nevertheless, also a configuration or a software component that runs on a virtual machine could be tampered to change its behaviour. Therefore, also these threats must be addressed for the correct management of these architectures.

In order to evaluate the trustworthiness of these systems, the Trusted Computing techniques must be considered. In particular, the Remote Attestation (RA) technique makes it possible to verify the integrity of binaries and their configurations, although with some limitations.

This paper builds upon a preliminary work [1, 2] to propose a novel formal model for the validation and analysis of security policies, and a new approach, based on Trusted Computing techniques, to evaluate the trustworthiness of the security applications deployed into the network to enforce the policies.

The paper is organized as follows: Section 2 presents the approach, Section 3 describes the model for monitoring the policy enforcement, Section 4 describes the model for policy conflict analysis, Section 5 defines the approach used to attest the trustworthiness of security controls, and Section 6 contains the implementation details of our prototype and the related results. Finally, Section 7 discusses some related works on policy monitoring, conflict analysis, and remote attestation.

## 2 Approach

In this paper, we present a unified approach to evaluate the enforcement of security requirements, expressed by a set of security policies. This evaluation enables the possibility to guarantee an adequate security level in the network, since functional configuration of each deployed security control is checked to be consistent with the defined requirements. The presented approach consists of on-line policy en-

forcement validation by using monitoring and remote attestation. In case of failure, an off-line detection of the causes is performed. Possible causes of failure in policy enforcement can be derived from: *(i)* errors in manual changes of the configuration rules for a security function, that can alter the application behaviour or introduce policy anomalies; *(ii)* untrustworthiness of the security functions deployed into the network (e.g. modifying a software binary code to modify its behaviour).

In order to perform a comprehensive monitoring of the enforced security controls' [1] configurations, first of all, we defined a High Level Security Policy Language (HSPL), useful to define security requirements by using an abstract and high-level approach. This clearly simplifies the policy authoring task for network administrators, because they can define the security requirements as a set of sentences close to natural language, e.g. "`do not access gambling sites`","`allow Internet traffic from 8:30 to 20:00 for employees`". In particular, the elements of a sentence (subject, object, etc.) can be selected by the administrator from a predefined set and implemented in an editor as different lists. This approach is transparent for administrators (avoiding them to learn a new language) and makes it possible to map each element of a sentence to the related HSPL component. This approach is also flexible enough to enable administrators to customize some elements of a sentence, e.g. to specify timing constraints or URLs.

Starting from a set of HSPL statements, we designed a policy validation framework, composed of several components, and its three-phases validation workflow. As depicted in Figure 1, two out of three phases are performed on-line (i.e. the involved processes work at run-time, when the network is enabled), while the third phase is performed off-line (i.e. it is triggered regardless of the network state). In particular, the framework includes:

- **Policy monitoring** to periodically check the enforced HSPLs. In case of wrong enforcement, the process triggers another service to perform further analysis to detect the causes;
- **Conflict analysis**, triggered by the monitoring service to perform an exhaustive detection of anomalies among the configuration rules installed into the security functions;
- **Remote attestation** to verify the trustworthiness of the network functions and the other components involved in the validation process.

As depicted in Figure 1, the monitoring process is performed by the *Monitor* component, which checks, by exploiting refinement techniques, if security functions are configured to enforce the HSPLs. This component relies on sender and receiver modules, deployed in the network. These nodes (depicted in Figure 1 respectively as "`S`" and "`R`") are transparent functions that generate, forward and monitor packets, without modifying them, and collect information for the Monitor.

When the Monitor detects wrong enforced HSPLs, it triggers the conflict analysis service to identify the causes of the misbehaviour. This service is implemented by another component, named *Analyser*, which exploits formal techniques for detecting anomalies of configurations. By means of First Order Logic and Boolean modeling, the Analyser is able to detect when an anomaly is triggered by one or more configuration rules. In case of anomaly, the administrator is notified by a report of the detected anomalies and related configuration rules.

The workflow also includes a third component, the *Verifier*, to attest the trustworthiness of the network by using remote attestation. The Verifier works simultaneously with the Monitor, as an on-line service. This component, thus, is in charge of verifying the trustworthiness of all the elements involved in the workflow, i.e. the network functions and the newly introduced components (and their sub-elements). If the remote attestation fails, the Verifier immediately sends an alert to the administrator, specifying the causes.

[1] In this paper, we use interchangeably the terms security controls (or controls) and security functions (or functions).

In particular, in our validation workflow (Figure 1), the Analyser is triggered only in case of wrong enforced HSPLs and positive attestation result. Therefore conflict analysis is not performed when there is at least one untrusted element. This choice has been proposed to avoid the overhead required by the analysis module and to increase the overall performance.

Having described an overview of the whole framework, we can move to present each of its components and their processes.

## 2.1 Policy Monitoring

The policy monitoring process is in charge of verifying enforced HSPLs. The Monitor, initially, receives the security requirements (expressed as set of HSPLs) and computes the set of network packets that should be generated to check the policy correctness. Then it configures sender and receiver nodes (named Senders and Receivers) by delegating to them a sub-set of packets per-security control.

In details, the Senders receive periodically a set of packet parameters from the Monitor and generate the network traffic for a specific security control. Depending on the type of security control, a sender node can generate different types of traffic (e.g. HTTP, SMTP, FTP). The Receivers, instead, are transparent functions, which observe the packets coming out by security controls to produce a summary for the Monitor. By this design choice, it is possible to place a Receiver directly into a security control or external to it[2]. On the other hand, when every Receivers summaries are collected, the Monitor analyses and checks the policy correctness: in case of errors, it triggers the components in charge of detecting the causes of the misconfiguration (i.e. the Analyser).

The definition of the network packets for a security control is based on a formal model that adopts the same approach of the refinement [3]. The input are the HSPLs and the output are the type of traffic that each Sender must generate and the expected traffic that each Receiver should collect from the associated security control. When the Monitor receives the summary, by the Receivers, it analyses whether a HSPLs is correctly enforced.

## 2.2 Conflict Analysis

Once the Monitor detects which policies are not enforced and, in turn, which controls are not correctly configured, the goal of conflict analysis is to identify whether the deployed configuration rules are anomalous. This process is performed offline, within the Analyser component (Figure 1).

As discussed before, security policies are described by using the HSPL statements, however security controls typically have low-level specific format. Therefore, in order to perform its goal, the Analyser has to collect and transform the installed configuration rules into an internal formalism. This formalism is an intermediate format between the HSPL and the low-level configuration rules: it is the Policy Implementation (PI) data-structure.

The Analyser exploits formal models for modelling both the configuration rules (expressed in form of the Policy Implementation data-structure) and the anomalies to check. In particular, anomalies are expressed in form of detection rules, which are FOL formulas that can be satisfied by one or more configuration rules. The goal of conflict analysis is in fact to check which detection rules are triggered by the received input (i.e., the PI set): each detection rule that is satisfied represents a conflict in the configuration rules installed in the network. At the end of the analysis, the end-user (e.g., network administrator) is alerted by a report, where the detected anomalies and the configuration rules that have triggered those anomalies are indicated.

[2]Senders and Receivers do not require any particular hardware resource (e.g., CPU, memory): Sender nodes need to support traffic generator tools such as Scapy, netsniff-ng or packet sender, while Receiver nodes must support monitoring tools like Wireshark, Packetyzer and Ostinato.

### 2.3 Remote Attestation

In this framework the remote attestation is implemented by the verifier component (hereafter Verifier) compliant with the trusted computing standards published by Trusted Computing Group (TCG)[3]. Remote attestation is able to provide hardware-based authentic evidence of the attesting platform (hereafter Attester) integrity state with the help of a special designed hardware chip called *Trusted Platform Module* (TPM), which has been installed in most business class laptops, desktops and servers. In virtualised environments, where direct access to the hardware chip is limited or missing, a virtualised TPM (vTPM) running in a difference virtual domain can be adopted to provide the same functionalities but with less security guarantees [4]. However, vTPM solution currently is only available in Xen[4].

The primary goal of the Verifier is to periodically attest the nodes hosting security applications in the framework and cooperates with the conflict analyser, which analyses anomalies of the applied policies only when security controls are in trusted state. Furthermore, it can even attest every other components (e.g. Senders, Receivers) to ensure the whole framework is in trusted state, implying the genuineness of the test result. The integrity state of a security control ensures that: *(i)* the system is booted with all known components in a predefined order, which implies the system is running in a trusted state without any bootkit attack; *(ii)* the services running in the application layer of each host are loaded with legitimate executables and known service configurations with reference to a well formed database, which implies the services are running in trusted state to deal with their inputs correctly. In case of compromised node, either because of remote attacks or wrong service configurations, the Verifier alerts the administrator of this integrity state change.

## 3 Policy Monitoring Service

In this section we start to describe more in depth the first component of our validation framework, that is the Monitor, which implements the policy monitoring in our solution. We recall that the monitoring process is performed by sending a set of packets (potentially also a single packet) to each security control that enforces a HSPL and checking if such controls forward or discard properly the probe-packets, with respect to the HSPL. This means that the Monitor needs a set of HSPL as input and it must implement a strategy to calculate the set of probe-packets to forward from the received HSPLs.

### 3.1 High Security Policy Language

As discussed before, the administrator specifies the security requirements with the HSPL. Starting from previous works [3], we designed HSPL as an authorization language that follows the *subject-action-object-attribute* paradigm (also referred to as *target-effect-condition*) [5]. More precisely, a generic HSPL $h$ of the whole set of HSPLs ($h \in \mathtt{H}$) is defined as the following n-tuple:

$$h = \{s, a, o, f\}$$

- $s$ is the subject of the policy, that can be a single user, a group or all the users (e.g. employees, guests, administrators);
- $a$ indicates an action that specifies whether a subject $s$ is (or not) authorized to access a resource, to send or to receive packets. In our model, this field takes as value: "`authorize access`", "`not authorize access`", "`authorize to send`", "`not authorize to send`", "`authorize to receive`" or "`not authorize to receive`";

[3] `https://www.trustedcomputinggroup.org`

[4] `http://wiki.xenproject.org/wiki/Virtual_Trusted_Platform_Module_(vTPM)`

- $o$ represents the object controlled by the action $a$ and it supports different types of traffic, that are "`Internet traffic`", "`DNS traffic`", "`VoIP traffic`" and "`all traffic`";
- $f$ is an optional set of conditions useful to specify the scope of an object (e.g. a specific URL).

With regard to the $f$ element of a HSPL, this is composed by a set of conditions defined on header fields of a packet, where each condition is a pair <*type, value*>. Thus, a generic $f$ has the following structure, where $t_n$ represents the type of the field and $v_n$ is its value:

$$f = \{\ (t_1, v_1), \ldots , (t_n, v_n)\}$$

The supported field types are: *(i)* `time` restricts the application of HSPL by imposing timing conditions (e.g. specific date or range, single day of the week, type of days like the weekend or vacation); *(ii)* `URI` limits the policy considering a specific URI or URL (e.g. www.facebook.com); *(iii)* `content type` defines specific types of content in the packet, such as illegal content, gambling, explicit sexual content, etc.; *(iv)* `traffic target` specifies a particular stakeholder that is generating the traffic flow. The possible stakeholders are defined by the end-user ( e.g. `corporate_network` or `user`).

For the sake of simplicity, in this paper, we have considered only HSPLs related to the traffic filtering and content inspection. This means that, in our validation framework in general and in our monitoring process in particular, we only use a subset of actions, objects and fields of the whole set supported by the HSPL language. However, the model can be easily extended to support the entire set of the HSPL.

### 3.2 Traffic flow generation model

We recall that the Monitor is in charge of configuring the Senders with a set of packets to generate. To represent traffic, our approach models a packet $p$ ($p \in \mathtt{P}$) by defining of a set of conditions imposed on the header of the packet. Such conditions are grouped considering network layers, following this structure:

$$p = \{\mathscr{L}_3, \mathscr{L}_4, \mathscr{L}_7\}$$

In details, $\mathscr{L}_3$ supports source/destination IP addresses and protocol type (e.g. IPv4). At layer 4 ($\mathscr{L}_4$), we consider the source and destination port numbers and protocol (e.g. 22/TCP). At application layer ($\mathscr{L}_7$), the model considers the application protocol (e.g. HTTP), method (e.g. GET) and URI (e.g. www.facebook.com). Thus the layers have the following structure:

$$\mathscr{L}_3 = \{\mathrm{IP}_{src}, \mathrm{IP}_{dst}, \mathrm{IP}_{prot}\} \qquad \mathscr{L}_4 = \{\mathrm{P}_{src}, \mathrm{P}_{dst}\} \qquad \mathscr{L}_7 = \{\mathrm{A}_{prot}, \mathrm{M}, \mathrm{U}\}$$

In order to generate the proper set of packets to check the well-enforcement of HSPLs, we compute the probing packets by using the function $v(h_i)$. The function receives a HSPL $h_i$ and outputs one or more packet-sets $e_i$. Each packet-set $e_i$ indicates a sub-set of the whole set of packets ($\mathtt{P}$) that can be generated in the network. Thus, $v(h_i)$ identifies the set of conditions to check the HSPL $h_i$ and it is defined as:

$$v(h_i) = \{e_1, e_2, \ldots, e_n\} \quad \text{where:} \quad e_i = \{p_1, p_2, \ldots, p_n\} \quad \forall e_i \subseteq \mathtt{P}$$

Let us consider an example of HSPL $h_1$ defined as *"The employees are not authorized to access Internet {(specific uri, www.facebook.com)}"*. The function $v(h_1)$ will identify the probe-packet $p_1$ structured as follows:

$$v(h_1) = \{e_1\} = \{(p_1)\}, \quad p_1 = (\mathscr{L}_7\{\mathrm{A}_{prot} = \mathrm{http}, \mathrm{U} = \text{www.facebook.com}\})$$

Another example is $h_2$ that is defined such as *"The employees are not authorized to access Internet {(type of content, social networks)}"*. Here, the function $v(h_2)$ identifies a set of $n$ packets, one for each website in the black list of social networks, i.e.:

$$v(h_2) = \{e_2\} = \{p_1, p_2, ..., p_n\}$$

In this paper, we denote the whole set of probe-packets as $\mathscr{V} = v(h_1), v(h_2), ..., v(h_n)$. Using the set cover problem, we compute the Minimal Hitting Sets (MHSs) of packets that correctly enforce all HSPLs. In particular, a set $s$ is said to be a Hitting Set (HS) of $\mathscr{V}$ if and only if:

$$HS(\mathtt{P}, \mathscr{V}, s) \leftrightarrow s \subseteq \mathtt{P} \wedge \forall_{v \in \mathscr{V}} \exists_{e \in v} : s \cap e \neq \emptyset$$

A set $s$ is said to be a Minimal Hitting Set (MHS) of $\mathscr{V}$ if and only if:

$$MHS(\mathtt{P}, \mathscr{V}, s) \leftrightarrow HS(\mathtt{P}, \mathscr{V}, s) \wedge \nexists s' \subset s : HS(\mathtt{P}, \mathscr{V}, s')$$

Even though, in literature several approaches to solve the MHS problems have been presented, we exploit the solution proposed in [6] within our monitoring service. After the computation of $MHS(\mathtt{P}, \mathscr{V}, s)$, according the network topology $\mathtt{N}$, we can configure Senders by generating the matrix $M_S$. This matrix indicates that the $j$-th Sender ($S_j$) must generate the probe-packet $p_{ij}$ to check the $i$-th HSPL ($h_i$). Thus $M_S$ is structured as follows[5]:

$$M_S = \begin{array}{c} \\ \mathbf{h_i} \end{array} \begin{array}{c} \begin{array}{ccc} & \mathbf{S_j} & \end{array} \\ \begin{bmatrix} p_{1,1}/0 & \cdots & p_{1,j}/0 \\ \vdots & \ddots & \vdots \\ p_{i,1}/0 & \cdots & p_{i,j}/0 \end{bmatrix} \end{array}, h_i \in \mathtt{H} \wedge S_j \in \mathtt{N}$$

### 3.3 HSPL enforcement validation

Before checking if the security controls really enforce the HSPLs, the Monitor has to configure the Receivers. In order to enable Receivers to collect data about incoming traffic, we predict how a security control should process (and forward) the probe-packets. This functional behaviour of a security control is modelled by means of a matrix. The matrix $M_{c_k}(t)$ represents how the security control $c_k$ processes the incoming packets at time $t$. The matrix $M_{c_k}$ is, in fact, time-dependent, because a security control can change its processing over time (e.g., administrator changes its configuration), and it is structured as follows:

$$M_{c_k}(t) = \begin{array}{c} \\ \mathbf{h_i} \end{array} \begin{array}{c} \begin{array}{ccc} & \mathbf{p_j} & \end{array} \\ \begin{bmatrix} \pm 1/0 & \cdots & \pm 1/0 \\ \vdots & \ddots & \vdots \\ \pm 1/0 & \cdots & \pm 1/0 \end{bmatrix} \end{array}, h_i \in \mathtt{H} \wedge p_j \in \mathtt{P}$$

The $ij$-th element of the matrix can take as value: 1, if the HSPL $h_i$ defines to forward the packet $p_j$ to the next hop (i.e., the Receiver associated to $c_k$); -1, if the HSPL $h_i$ defines to block the packet $p_j$ or $h_i$ is not supported by the security control $c_k$; 0, if the HSPL $h_i$ is not related to the packet $p_j$.

In order to reduce the complexity of checking if the security control is correctly enforcing the HSPLs, we calculate the *equivalent policy* of the security control $c_k$ from its behavioural matrix $M_{c_k}(t)$. The equivalent policy is the result of a resolution strategy applied on $M_{c_k}$. The resolution strategy adopted in this paper is the First Order Matching. Thus, the equivalent policy of a security control is expressed by

[5]Conventionally, we put a zero value in the position $ij$ if the Sender $S_j$ must not generate a packet to verify the HSPL $h_i$.

an array $A_{c_k}(t)$ structured as follows, where the $j$-th element is equal to the first non-null element of $M_{c_k}$ in the $j$-th column:

$$A_{c_k}(t) = \begin{matrix} & \mathbf{p_j} & \\ \big[ & {}^{-1}/_{1} \quad \cdots \quad {}^{-1}/_{1} & \big], p_j \in \mathtt{P} \end{matrix}$$

We denote with $M_c$ the matrix that collects the $A_{c_k}$ for any security control $c_k$. The $M_c$ matrix, thus, summarizes, the packet forwarding and processing of the whole network and it is defined as follows:

$$M_c(t) = \begin{bmatrix} A_{c_1}(t) \\ \vdots \\ A_{c_n}(t) \end{bmatrix} = \begin{matrix} \mathbf{p_j} \\ \begin{bmatrix} {}^{-1}/_{1} & \cdots & {}^{-1}/_{1} \\ \vdots & \vdots & \vdots \\ {}^{-1}/_{1} & \cdots & {}^{-1}/_{1} \end{bmatrix} \end{matrix}$$

Having modelled the expected forwarding and processing behaviour of a security control, we can describe how Receivers collect information about the received probe-packets from the associated controls. In particular, we recall that a Receiver is associated to one or more security controls and for each of them it checks if the controls are forwarding (or dropping) the probe-packets as the Receiver has expected by means of the equivalence policy $A_{c_k}$ of that control. A Receiver $r_k$, thus, collects information about the received packets in the time interval $T$ in an array $A_{r_k}(T)$. This array is structured as follows, where the $j$-th element can take as value 1, if the packet $p_j$ has been forwarded to the Receiver $r_k$ during the interval $T$, otherwise it is -1:

$$A_{r_k}(T) = \begin{matrix} & \mathbf{p_j} & \\ \big[ & {}^{1}/_{-1} \quad \cdots \quad {}^{1}/_{-1} & \big], p_j \in \mathtt{P} \end{matrix}$$

We denote with $M_r$ the matrix that collects the $A_{r_k}$ for any Receiver $r_k$. The $M_r$ matrix, thus, summarizes, the data collected by every Receiver in the network and it is defined as follows:

$$M_r(T) = \begin{bmatrix} A_{r_1} \\ \vdots \\ A_{r_n} \end{bmatrix} = \begin{matrix} \mathbf{p_j} \\ \begin{bmatrix} {}^{1}/_{-1} & \cdots & {}^{1}/_{-1} \\ \vdots & \vdots & \vdots \\ {}^{1}/_{-1} & \cdots & {}^{1}/_{-1} \end{bmatrix} \end{matrix}$$

Finally the validation of the enforced HSPLs in the interval $T$ is performed by the Monitor by doing the difference between $M_c$ and $M_r$. If the Monitor returns a null result, this means that the HSPLs have been correctly enforced, otherwise there is at least one non-well enforced HSPL in the network.

Furthermore, thanks to the proposed matrix-based model,t is possible to check whether a security control is not correctly configured and which packet is not correctly processed. In this case, a further analysis is required in order to detect the causes of such misconfiguration. The next section presents the component in charge of performing a conflict analysis.

# 4 Conflict Analysis Service

We now move to present the conflict analysis approach, by focusing on how the installed configuration rules are modelled and how the anomalies are detected against those rules. We recall that the Analyser is triggered by the Monitor , only in case the Verifier does not detect any untrusted state in the network. When the Analyser, instead, detects some anomalous configuration that lead to conflicts, it alerts the administrator indicating the configuration rules that trigger an anomaly and also the triggered anomalies.

### 4.1 Policy Implementation Model

The Analyser relies on a formal model for representing both the configuration rules and the anomalies. In order to perform a conflict detection, we have identified four elements to distinguish traffic flows and all the required information to describe a network behaviour and the network conditions under which an anomaly arises. The analysis model, thus, includes the following elements, which are:

- *network fields*, atomic elements to identify the main data that a configuration rule of a security control should track, like packet headers, network node ID, traffic label, cipher algorithm and many other that depends on the particular instance of the security control (i.e, firewall rather than deep packet inspection);
- *policy actions*, atomic elements that represent a real action performed by a security control and all the information that characterized it (e.g., algorithm, technologies, protocols to use). For example a firewall that is configured to perform a `drop` action in case a packet matches with a firewall rule;
- *Policy Implementations* (*PIs*), data-structures to pinpoint in a formal and abstract way the configuration rules enforced by a security control. In particular, *PI* has a structure composed of a sequential set of network fields ($n$) and a set of policy actions ($a$). From now on, we use the notation $n_{in}$ to indicate the $n$-th network field in the $i$-th PI $pi_i$, while $a_{im}$ is the $m$-th action of $pi_i$. Hence a PI $pi_i$ can be represented as follows:

$$pi_i = (n_{i1}, n_{i2}, ..., n_{in}, a_{i1}, a_{i2}, ..., a_{im})$$

- *Detection Rules* (*DRs*), sets of conditions that distinguish the possible anomalies among *PIs*. The model is flexible enough to support a set of pre-defined detection rules, that have been indicated by the literature as well-known anomalies in the policy analysis domain, but also it allows administrators to define their own set of anomalies, and, in turn, detection rules.

### 4.2 Network Field and Policy Action Relations

For what concerns the detection of anomalies against the *PIs*, the model needs to support a set of relations $\mathfrak{R}$ among the network fields and policy actions of the *PIs* to define the detection rules to be checked (i.e., to detect the anomalies). In detail, the proposed model supports four relations $\mathfrak{R}$ between network fields:

- **equivalence (=)** two network fields are equivalent (or equal) if they have exactly the same value (e.g., $n_{ik} = 8080 \wedge n_{jk} = 8080 \implies n_{ik} = n_{jk}$);
- **dominance** ($\succ$): a network field dominates another one if it is a generalization of the latter. For example, let us consider two IP addresses $n_{ik}$ and $n_{jk}$, if $n_{ik} = 1.1.*.*$ and $n_{jk} = 1.1.1.*$, then $n_{ik} \succ n_{jk}$;
- **correlation** ($\sim$): two network fields are correlated if they share some common values, but none of them includes (or dominates) the other one. For example, if $n_{ik}$ and $n_{jk}$ are ranges of port numbers and $n_{ik} = [1,75]$, while $n_{jk} = [50,100]$, then $n_{ij} \sim n_{jk}$ because the range [50, 75] is shared by both fields;
- **disjointness** ($\perp$): two network fields are disjoint if they do not share any value. Considering the previous example about ranges of port numbers, if $n_{ik} = [1,70]$ and $n_{jk} = [71,100]$, then $n_{ik} \perp n_{jk}$. On the other hand, if a network field is equivalent, correlated or dominates another one, those fields are *non-disjointed* ($n_{ik} \not\perp n_{jk}$).

A certain relation can be applied to a network field (or a policy action) depending on the type of the network field value (e.g., integer, IP address, boolean, enumeration and more). For example, a port number can be equal to another one, but it cannot dominate a range of port numbers. Instead a range of port numbers can dominate a single value of port number. With respect to the policy actions of a *PI*, the same relations can be defined for the actions.

The proposed set of relations allows a flexible definition of conditions on the *PI* elements, of which the value types are not known a-priori. In fact when the *PI* structure is established for a certain security control and, in turn, the network field types, it is possible to understand which relation $\Re$ is supported by a field (or action). In this way, the model achieves high-level flexibility and generality in order to: *(i)* not be limited to a single security control; and *(ii)* enrich the set of anomalies by allowing administrators to define their own set.

### 4.3 Policy Anomaly and Detection Rules

Policy anomalies ($A$) are defined in form of *detection rules*, which, in turn, are expressed by a set of relations $\Re$ among the network fields and policy actions. In particular, a detection rule can contain conditions that involve network fields and actions of one or more *PIs*.

We model a detection rule as a First Order Logic (FOL) formula, expressed using Horn clauses. Horn clauses are frequently encountered in model theory because they exhibit a simple and natural rule-like form. Also, these clauses can be easily translated into many different logic programming languages, such as Prolog, or generic programming language such as C or Java. In particular, Horn clauses can be simply used to represent all the axioms used in the proposed model, expressed in the form of positive conditions implying an assertion:

$$C_1 \wedge C_2 \wedge \cdots \wedge C_n \Rightarrow A$$

In our model, also, every clause is the relation between network fields (e.g., $n_k$) or policy actions (e.g., $a_k$) of one or more *PIs* (e.g., $pi_i$ and $pi_j$), like this example:

$$C_1 := n_{ik} \,\Re\, n_{ih}, \quad C_2 := n_{ik} \,\Re\, n_{jh}, \quad C_3 := a_{ik} \,\Re\, a_{ih}, \quad C_4 := a_{ik} \,\Re\, a_{jh}$$

where, we recall, $n_{ik}$ and $n_{jh}$ (or $a_{ik}$ and $a_{jh}$) identify two generic network fields (or policy actions) in the *PI* structure. Hence possible detection rules can take a form like the following, but they are not limited to this structure:

$$n_{iq} \,\Re\, n_{jq} \wedge \ldots \wedge n_{ik} \,\Re\, n_{jh} \wedge \ldots \wedge a_{ik} \,\Re\, a_{jh} \Rightarrow A$$

In order to simplify the understanding of this model (i.e, its elements and the supported relations) and how to define detection rules, we can consider the filtering configuration rules that can be installed in a generic firewall as use case, which is presented in the next section.

### 4.4 Filtering Policies Use Case

Filtering rules are generally used in a single- or multi-firewall environment to defend networks by filtering unwanted or unauthorized traffic from or to the secured network. In order to define the PI model for representing filtering rules, we have extrapolated the main network fields and actions needed to distinguish the filtering conflicts that may arise in firewall configurations.

In particular, we have considered as PI elements: a firewall identifier ($f$) to distinguish in which order a packet is processed by a firewalls chain in the network; a filtering rule identifier ($r$), valid within the firewall $f$; the source and destination IP addresses of the traffic ($ip\_src$ and $ip\_dst$); the protocol

type $t$ (we have considered a sub-set of the possible protocol types, that is `TCP`, `UDP` and `* - don't care`); source and destination port number ranges ($p\_src$ and $p\_dst$); and finally the filtering action ($a$) performed by the firewall $f$ in case the received packet matches with the current set of fields (we consider `accept` or `deny` as possible action value). Hence we have structured a filtering PI as:

$$pi_{fp} = (f, r, ip\_src, ip\_dst, t, p\_src, p\_dst, a)$$

In this use case, the relations $\Re$ supported by the aforementioned filtering *PI* structure are: $f_i$ and $r_i$ in $pi_i$ can be equal to or can dominate their counterpart in $pi_j$ (e.g., $f_i \succ f_j$, if $f_i$ is equal to 6, while $f_j$ is equal to 3); $ip\_src_i$, $ip\_dst_i$, $p\_src_i$ and $p\_dst_i$ can be equal, disjointed or can dominate the relative fields of $pi_j$; finally, $a_i$ and $t_i$ can be equal or disjointed respectively to $a_j$ and $t_j$.

**Filtering anomalies** As example of detection rules for our use case, we can consider the anomaly classification proposed in [7]. In this case, we show the detection rules of two examples of anomalies that the Analyser can detect. Those detection rules follow the aforementioned definition of filtering *PI* and are:

- *Intra-Firewall Shadowing anomaly* arise when $pi_i$ and $pi_j$ match the same traffic, but they enforce different actions (e.g., one PI drops the traffic, while the other policy allows it):

$$f_i = f_j \wedge r_j \succ r_i \wedge ip\_src_i \succeq ip\_src_j \wedge ip\_dst_i \succeq ip\_dst_j \wedge t_i \succeq t_j$$
$$p\_src_i \succeq p\_src_j \wedge p\_dst_i \succeq p\_dst_j \wedge a_i \neq a_j \Rightarrow \text{Shadowing}(pi_i, pi_j)$$

- *Inter-Firewall Redundancy anomaly* occurs when two *PIs* installed in different firewalls have the same filtering behaviour, This means that they match the same traffic and apply same action of blocking to it:

$$f_j \succ f_i \wedge ip\_src_i \succeq ip\_src_j \wedge ip\_dst_i \succeq ip\_dst_j \wedge t_i \succeq t_j \wedge \quad (1)$$
$$p\_src_i \succeq p\_src_j \wedge p\_dst_i \succeq p\_dst_j \wedge a_i = a_j = \text{deny} \Rightarrow \text{Redundacy}(pi_i, pi_j)$$

It is interesting noting that only the administrator can decide which anomaly is a real unwanted network behaviour, and when it is needed to re-adapt the set of filtering anomalies for his network. Another interesting feature of the proposed framework could be an automatic resolution of the detected anomalies by applying administrator-defined decision strategies. However we leave this interesting topic to be investigated as future work.

Next section focuses on the last part of the framework, that is the Verifier. This component takes care of checking periodically the trustworthiness of the network and, in case of failure, it alerts the administrator.

# 5 Remote Attestation Service

This section presents the details of remote attestation technique, in order to allow readers to understand its functionality and how it is used in our policy validation framework.

## 5.1 Background

Remote attestation is the main feature provided in trusted computing technology. The overall scheme proposed by TCG for using trusted computing is based on a step-by-step extension of trust, called a

*chain of trust*. It uses a transitive mechanism: if the first execution step can be trusted and each step correctly measures the next executable software for integrity, then the overall system integrity can be evaluated. Thus, during the loading of each piece of software, the content of each piece of software is measured and stored inside a log. Later, at the request of an external party, the attester can present this log, signed with a unique asymmetric key of the platform, to the other party to prove the platform identity and integrity state of platform.

From the chain of trust extension point of view, any component that needs to be loaded is considered an adversary and it must be measured before it is loaded. The base case for the extension of the chain of trust is called the *root of trust for measurement*, it encompasses the minimal combination of hardware and software elements that a remote verifier needs to trust in order to validate the entire chain of trust. It is recommended to use hardware device in combination with software components to create a strong unforgeable identity and provide safer storage of evidence. Therefore the main components of the Root of Trust for Measurement are: (i) a specialised hardware component to store the log and measurements away from the software access, (ii) an initial isolated component that is able to measure the first non–trusted software, which will be then trusted to measure the next stage software.

When using a TPM as root of trust, measures of the software stack are stored in special on-board *Platform Configuration Registers* (PCRs). There are normally a small number of PCRs (at least 16) that can be used for storing measurements. For security reasons, it is not possible to directly write to a PCR; instead measures must be stored using a special operation called *extend*. The extend operation can update a PCR by producing a global hash of the concatenated values of the previous PCR value with the new measurement value, such as the following:

$$\text{PCRnew} = \text{SHA-1}(\text{PCRold} \,||\, \text{measurement})$$

This approach brings two benefits. First, it allows for an unlimited number of measures to be captured in a single PCR, since the size of the values is always the same and it retains a verifiable ordered chain of all the previous measures. Second, it is computationally infeasible for an attacker to calculate two different hashes that will match the same resulting value of a PCR extend operation.

Besides strong isolated storage, the TPM also provides a unique key whose private part never leaves the TPM. This key is called *Endorsement Key* (EK) that is injected when the TPM is manufactured. To preserve the privacy of a platform identity, *Attestation Identity Keys* (AIKs) are generated and used in remote attestation process instead of the EK. The AIK is an alias of EK, whose private part never leaves the TPM which generates it. However, binding the EK and the AIKs of a TPM must be done in conjunction with a third party *Privacy Certificate Authority* (PrivacyCA) who is trusted to not reveal the real platform identity, but to act as an intermediary.

When a platform receives a remote attestation request, it needs to send back an integrity report which comprises the values stored in the PCRs and their digital signature computed with an AIK. Since the private parts of the AIK is never released from the TPM, thus the authenticity and integrity of the integrity reports are guaranteed.

To be more specific, the operation to get signed PCR values from a TPM is called *quote*. This operation is simple from both the verifier's and the attester's point of view. The verifier wishing to validate the integrity state of the attester sends a remote attestation request specifying an AIK for generating the digital signature, the set of PCRs to quote, and a nonce to ensure the freshness of the digital signature.

After the TPM receives the remote attestation request, it validates the authorisation to use the AIK, fills in a structure with the set of PCRs to be quoted and generates a digital signature on the filled in structure with the specified AIK. Then it returns the digital signature to the verifier.

In general, the verifier, after receiving the digital signature, validates the integrity of the PCR values received using the public portion of the AIK. Then if the PCR values are intact, then the verifier assesses

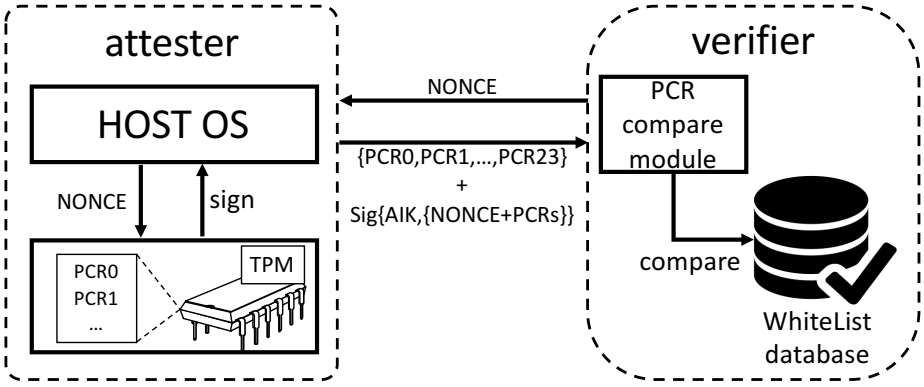



Figure 2: Remote attestation process.

the attester's trustworthiness in reference to a whitelist database (Fig. 2).

### 5.2 Trusted boot

Trusted boot is used to ensure the platform is booted into a trusted state, it can be achieved by storing digests of the component loaded in booting phase into different PCRs. The following list describes which PCR registers can be used during a trusted boot process:

- PCRs 00-03: for use by the CRTM (Initial EEPROM or PC BIOS)
- PCRs 04-07: for use by the Bootloader stages
- PCRs 08-15: for use by the booted base system (e.g. compartmentalisation system, hypervisor)

As shown above, the PCR values are fixed if and only if the same components are loaded in a pre-fixed order. Thus, when a verifier receives a set of authentic PCR values, it can be sure the platform is booted by loading trusted components, and the operating system is running in a trusted state.

### 5.3 Service measurement

Once the operating system is booted into a trusted state, it further measures each application loaded through a measurement module such as *Linux Integrity Measurement Architecture* (IMA) [8]. These measures are stored in a Stored Measurement Log (Fig. 4), with each one extended into a PCR in the TPM (Fig. 3). Thus the integrity of the log is implicit authenticated by the TPM. Later, the measures in log will be used during remote attestation protocol, as evidence to prove the integrity state of the loaded services.

For instance in Fig. 4, we chose four IMA measures of the most concern to a security application host as an example. `/usr/sbin/iptables` is the iptables executable loaded by the system kernel. `/etc/iptables-initial` is the initial configuration of iptables when the system is booted. SSH is used to access the security application host remotely, with `/usr/sbin/sshd` is the executable and `/etc/ssh/sshd_config` is the configuration file. The filedata-hash column shows the hash value (i.e. measure) of the file, it will be extended into PCR 10 in the TPM in a way illustrated in Fig. 3.

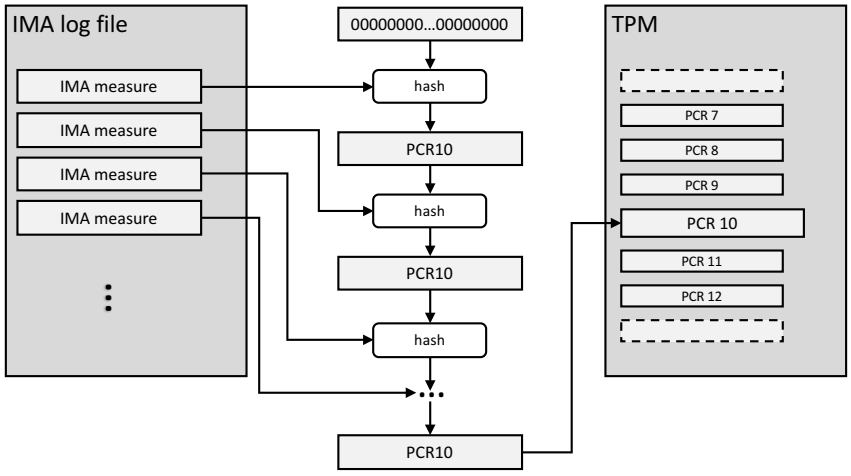



Figure 3: Extend operations for IMA measures.

```
PCR# template-hash template filedata-hash    filename-hint
 10  fc465...848ee   ima     e4092...732e6c   /usr/sbin/iptables
 10  48327...9fed4   ima     f7655...43f45c   /etc/iptables-initial
 10  bd57b...e45b3   ima     810cf...f821d6   /usr/sbin/sshd
 10  94ea2...eff6b   ima     960f7...a9728a   /etc/ssh/sshd_config
```

Figure 4: Example of IMA measurements in ASCII format.

### 5.4 Verification

When an integrity report is received by the verifier, the verifier first checks the digital signature of the report with the public part of the AIK. Then it compares the received PCR values to a whitelist database in order to check the boot phase of the attester is trusted. Afterwards, the verifier distils the IMA measures from the integrity report, and recomputes the final value following the extend operations illustrated in Fig. 3. If the final value equals the PCR value in the received integrity report, this proves the IMA measurement list is intact. Finally, the verifier queries the IMA measures to a well-formed database with whitelisted custom configurations. In the case that the received PCR value does not match the whitelist or there is an unknown measure, the verifier can alert the system administrator that the node is not booted in a trusted state or certain service in the application layer is compromised.

## 6 Implementation and Result

In order to validate our approach, we have implemented a prototype for each involved service. In this section, we present the implementation details and the results achieved in the evaluation phase, performed in terms of validation time under different network scenarios.

**Policy Monitor** We implement the Sender and Receiver nodes by using Scapy[6], a tool to generate network packets. More precisely, the Monitor instructs each Sender by passing the Scapy commands to

[6] http://www.secdev.org/projects/scapy/

generate the required packets. Then, each Receiver sends to the Monitor the summary of the received packets (captured by using Scapy).

The formal models exploited by the Monitor are developed by using Java 1.7 and several Java-based open-source frameworks, which are: *(i)* Drools[7], an expert system based on a rule-based approach, where a single rule is composed of conditions (checked against the input data) and consequential actions (performed when the conditions are matched); *(ii)* MOEA[8], a Java library for developing multi-objective evolutionary algorithms; *(iii)* the Apache Commons Mathematics Library (ACML)[9].

In particular, the Monitor exploits Drools and MOEA to infer and optimize the set of packets that matches (i.e. verifies) each HSPL policy, while the ACML is used to program Receiver nodes. Here, we report an example of Drools rule (Listing 1) for generating DNS packets (i.e. the action part of the rule), in case of an HSPL that refers to DNS traffic (i.e. the conditional part of the rule):

```
rule "DNS_traffic"
when
h: Hspl(h.getObject.compareTo(DNS_TRAFFIC)==0)
s: SetPacket()
then
new p Packet()
p.setTransport_protocol("UDP");
p.setDestinationPort(53);
p.setApplication_protocol("DNS");
s.addPacket(p);
end
```

Listing 1: example of Drools rule

**Conflict Analyser** The prototype implementation of the conflict analysis is based on an ontology, by exploiting the OWL2 language[10]. In particular, the Analyser has an OWL2 *class* for each network field and policy action of the *PI* and one for representing the *PI*. The relation among network fields or actions is instead represented by a set of *object property* assertions that connect the different classes of that network fields (or actions).

For what concern the *PI* anomalies, these are established by defining the *object property* assertion that links the *PI classes* involved into the anomaly. Then the detection rule that triggers that *PI* anomaly is expressed in the SWRL language [11]. We took this design choise because the SWRL language allows us to specify detection rules in a way similar to the Horn clauses.

Let us consider the example of the *PI* anomaly defined in formula 1. In this case we are considering the case of a Redundancy anomaly where two firewall rules match the same traffic and enforce the same action. This kind of anomaly can be detected by a detection rule, expressed in the SWRL language as follows (Listing 2).

In the first part of the rule, we specify a set of conditions to check if a OWL object (either a network field or an action), belongs to a policy implementation (e.g., `inFw(?pi1,?f1)` checks that the PI `pi1` contains the firewall identifier `f1`). The second part of the rule checks the relations among those fields and actions, like `Dom(?f1,?f2)` that returns true if the firewall identifier `f1` dominates `f2`.

[7] http://www.drools.org/
[8] http://moeaframework.org/
[9] http://commons.apache.org/proper/commons-math/
[10] https://www.w3.org/TR/owl2-profiles
[11] https://www.w3.org/Submission/SWRL/

```
inFw(?pi1,?f1),hasSrc(?pi1,?s1),hasDst(?pi1,?d1), hasType(?pi1,?t1),hasPSrc(?pi1,?ps1),
hasPDst(?pi1,?pd1),hasAct(?pi1,?a1),inFw(?pi2,?f2),hasSrc(?pi2,?s2),hasDst(?pi2,?d2),
hasType(?pi2,?t2),hasPSrc(?pi2,?ps2),hasPDst(?pi2,?pd2), hasAct(?pi2,?a2),
Dom(?f1,?f2),DomEq(?s1,?s2),DomEq(?d1,?d2),DomEq(?t1,?t2),DomEq(?ps1,?ps2),
DomEq(?pd1,?pd2),Eq(?a1,?a2),Eq(?a1,Deny)->Redundancy(?pi1)
```

Listing 2: example of SWRL rule

**Remote Attestation Verifier** The remote attestation framework used is *OpenAttestation SDK* (OAT)[12], an open source project initiated by Intel. We largely extended it to include IMA measures in the integrity report by following the specification released by TCG. Meanwhile, in order to verify the received IMA measures, we used our previous work [9], a set of python scripts to query the IMA measures to a well-formed cassandra[13] database.

When the verifier starts a remote attestation process (RA), the following six steps must be performed before the verdict can be issued (Fig. 5).

In order to minimise performance loss and attack surface of the attesting platform, the RA agent of OAT does not accept incoming attestation requests from third parties, rather it periodically polls the verifier (specifically registered at setup for this task) at a predefined interval (step 1). Once a RA request is present, the agent issues a *quote* operation to the TPM (step 2) for getting the PCR values and uses the result plus the recorded IMA measures to create the IR (step 3). The time to perform a quote operation is about 2 s while the time to prepare the IR is proportional to the number of IMA measures because each measure must be encoded in base64 and then appended to the XML template defined in the TCG specifications. Thus the size of the IR is also proportional to the number of IMA measures and so is the time needed to transmit the IR from the attester to the verifier (step 4). When the IR is received, the verifier checks the digital signature of the quote output against the public-key certificate of the attester, stored when the host machine was registered. Next the consistency of the IMA measurement list is checked against the PCR values, as previously shown in Fig. 3 (step 5). When these steps are completed, the measurement list is passed to the IMA verification script to verify them against the reference database (step 6). At this point, the verifier is able to return the integrity verification result.

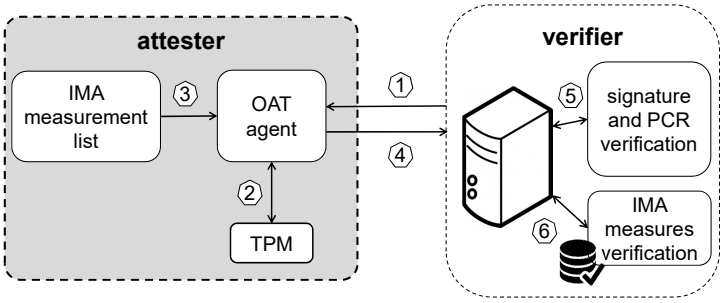



Figure 5: Overview of the operations in a remote attestation request.

## 6.1 Implementation results

To perform our test, we have instantiated our Monitor and Conflict Analyser prototypes in a workstation equipped with Intel i7-3630QM @ 2.4 GHz and 16 GB RAM.

[12] https://github.com/OpenAttestation/OpenAttestation/tree/v1.7
[13] http://cassandra.apache.org/

Each test was run on several network scenarios, which have been automatically generated. These scenarios differ for number of HSPLs and security controls (i.e., from 1 to 10), and, in turn, a variable number of configuration rules. Also we have adopted security controls that implement packet filters (i.e., netfilter/iptables) and application layer firewall (i.e., squid). The achieved data from these scenarios have been averaged over 50 test-runs.

With respect to the HSPL enforcement monitoring, we have performed two kinds of tests. First, we have kept the number of security controls fixed, increasing the number of HSPL policies (ranged from 1 to 50). Then, we have increase the number of security controls from 1 to 10, keeping constant number of HSPLs. In both cases, the Monitor achieves reasonable validation time, because it is able to check the well enforcement of 50 HSPLs into 10 security controls in the order of seconds (i.e., about 5 seconds).

For what concerns conflict analysis, even though it is a complex task, our Analyser prototype has achieved scalable results in detecting anomalies among filtering configuration rules. In particular, in the generated network scenarios we use, a variable number of configuration rules (derived from the HSPLs). This means that the Analyser has to manage a variable number of Policy Implementations (PIs), that was ranged from 50 to 500, 40% of which trigger an anomaly. Under these conditions, the Analyser has achieved a reasonable detection time of no more than 45s for checking 500 PIs.

Finally, the global performance of remote attestation process in concern mainly is the time needed to accomplish one remote attestation request. As described in Sec. 6, the most influential part is the time to prepare an integrity report in the attester and to verify the IMA measures to a reference database in the verifier. Both of them grow linearly with the number of IMA measures. Fortunately, because the security application host only needs a limited set of services, the number of IMA measure is only several hundreds, e.g. with CentOS7 distribution kernel 3.10.0-123.20.1.el7, the number of IMA measure is around 300. With this number, the time needed to prepare the integrity report costs about 3.5 s (including the time to get quote output from the TPM, which is about 2 second). And the time needed to query the IMA measures to a reference database hosted in the same machine costs about 1 second. Hence, the overall time required to attest a single security control is about 4.5 s. In order to test the performance in an intense scenario, we created a node which has 4500 IMA measures. The test result shows the time spent in the attester is 6.56 s and the time spent in the verifier is 2.35 s. Thus the overall time is 8.91 s that we deem a good result weighting the integrity guarantee provided by our solution.

In the case of attesting multiple security controls simultaneously, the total time should be smaller than the time needed to attest each one of them sequentially. Thanks to the parallel mechanism adopted, the time needed for getting quote output from each node is overlapped, thus improving the efficiency of attesting multiple nodes.

# 7 Related Work

This section gives an overview of the current state of the art about the three services available in the proposed framework. In particular, we present some of the approaches related to policy monitor and conflict analysis, with special focus on the considered use case (i.e., filtering policies). with regard to remote attestation, this section should help the reader to better understand proposed attestation approaches.

**Policy Monitoring** Firewall monitoring generally falls into the category of firewall testing. It consists in generating a set of packets to evaluate firewall decisions that are known a priori. If the firewall decision is the same decision as it was expected, the firewall configuration is correct, otherwise there are some errors. In literature many works are based on a firewall testing approach with different testing methods.

Hoffman [10] presented a framework to generate packet streams by using covering arrays, production grammars, and replay of captured TCP traffic. This framework, namely “Blowtorch” also supports packet

timing, traffic capture and replay and it is useful to check handshaking.

Jürjens [11] proposed a method for specification-based testing, enabling to formally model a firewall, surrounding network and to mechanically derive test cases to check for vulnerabilities.

El-Atawy [12] introduced a firewall testing technique by using policy-based segmentation of the traffic address space. This can adapt the traffic generation test considering potential erroneous regions in the firewall input space.

Senn [13] presented an approach to test the conformance of firewalls to a given security policy. The main contributions are: a language for the formal specification of network security policies, the combination of different methods for generating abstract test cases, and an algorithm for generating concrete test cases from the policy.

**Conflict Analysis** Concerning conflict analysis in the context of filtering domain, the main work was presented by Al-Shaer *et al.* who focus on anomalies occurring in a distributed firewall [7]. In spite of our approach, authors treat anomalies between pairs of rules, overlooking the case of anomalies generated by one or more than two rules. Moreover, they perform the anomaly detection by checking the packet header fields, without dealing with external information (e.g., algorithms, timestamps, etc...).

Khakpour *et al.* [14] have proposed a completely different approach to detect filtering anomalies based on a query engine system. However, this query-based approach does not find all the anomalies, as it checks only the scenarios specified by the administrator. In this approach, the impact of the human factor is very significant in the selection of the correct set of queries: an anomaly can be overlooked because it was not selected.

Other works, which take Al-Shaer *et al.* works as reference, were presented in literature. Basile *et al.* [15, 16, 17] extended the Al-Shaer's conflict classification, by relying on a geometrical representation of the model. The set of all packet header fields (namely the "selectors") generate an hyper-space, where the policy rules can be represented. Here, an anomaly is seen as the intersection of the space of two or more rules. Even though the authors fill the limitations of the Al-Shaer's work, they do not represent the anomalies generated by a single rule.

On the other hand, Garcia-Alfaro *et al.* [18] detect and resolve filtering anomalies thought the use of specific algorithms in distributed system, where NIDSs, VPN routers and other security controls are considered. However this is a radical different approach from our model, since rule-based inferential systems (formal ontologies, rule-based systems, SMT solver, etc...) do not use algorithms to detect anomalies.

**Remote attestation** Remote attestation has gained a lot of attention recently with the growing popularity of distributed systems. Taking advantage of the features provided by the hardware TPM, more advanced remote attestation techniques have been proposed in literature.

Property-based attestation was initially proposed by Sadeghi et al. [19] and generalised by Poritz [20]. The authors introduced property profiles to be mapped to system/service configurations with the help of proof-carrying code, to determine a set of specific properties is achieved by a set of specific configurations. Model-based behavioural attestation was proposed by Li et al. [21] and later generalised by Alam et al. [22]. In this approach, an additional component is required to collect system behaviour measures (e.g. as part of *Linux Security Module*) and extend them into the TPM. Afterwards, the verifier re-generates the system behaviour model and confronts it against a policy to assess the attester's trustworthiness.

One of the most influential work to enable remote attestation in virtualisation environment was proposed by Berger et al. [4]: it permits each virtual machine to have its own virtual TPM instance by simulating the hardware TPM's functionality. Along the same vein, Goldman et al. [23] adopted the

previous solution to manage a large amount of virtual machines. However, these two solutions cannot provide the same strong integrity guarantees as a hardware TPM. Alternatively, Garfinkel et al. [24] and Schiffman et al. [25] tried to modify the hypervisor to monitor the internal behaviour of virtual machines. Although interesrting, this approach brings significant performance loss and the direct link to the hardware TPM is still missing.

# 8 Conclusions

In this paper we have proposed a unified framework to validate network policy enforcement in a production environment by exploiting formal methods and remote attestation techniques.

The approach is extremely useful in many ways: for example, it can be used to detect errors and anomaly both in firewall implementation and firewall management. Our contribution is a formal approaches to: (1) identify/generate the traffic flow and to verify the enforced high-level policies; and (2) detect the causes of wrong enforced policies. We have designed and validated our approach by implementing a prototype of the whole validation framework, that is composed of different components (i.e. Monitor, Verifier and Analyser). The experimental results demonstrate that the approach is effective and has good performance, therefore our model can be effectively used to analyse real infrastructures. Currently, the approach has been implemented only for a limited set of policies, mainly related to filtering requirements (stateful packet filters and L7 filters). However, for the future, we plan to extend the expressivity of our model by adding the support for new types of security controls, such as VPN, proxy, IDS and IPS. Furthermore, we are planning to perform other empirical assessments in order to evaluate whether our tool can help administrators to reduce the number of anomalies in real-world scenarios.

### Acknowledgments

The research described in this paper is part of the SHIELD project, co-funded by the European Commission (H2020 grant agreement no. 700199).

# References

[1] F. Valenza, M. Vallini, and A. Lioy, "Online and Offline Security Policy Assessment," in *Proc. of the 8th ACM CCS International Workshop on Managing Insider Security Threats (MIST'16), Vienna, Austria*. ACM, October 2016, pp. 101–104.

[2] F. Valenza, S. Spinoso, C. Basile, R. Sisto, and A. Lioy, "A formal model of network policy analysis," in *Proc. of the 1st IEEE International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI'15), Torino, Italy*. IEEE, September 2015, pp. 516–522.

[3] C. Basile, A. Lioy, C. Pitscheider, F. Valenza, and M. Vallini, "A novel approach for integrating security policy enforcement with dynamic network virtualization," in *Proc. of the 1st IEEE Confernece on Network Softwarization (Netsoft'15), London, United Kingdom*. IEEE, April 2015, pp. 1–5.

[4] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, "vTPM: Virtualizing the Trusted Platform Module," in *Proc. of the 15th ACM Symposium on USENIX Security (USENIX'06), Vancouver, Canada*. ACM, July 2006, pp. 305–320.

[5] S. Godik, A. Anderson, B. Parducci, E. Damiani, P. Samarati, P. Humenn, and S. Vajjhala, "eXtensible Access Control Markup Language (XACML) Version 3.0," January 2013, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml [Online; Accessed on March 1, 2017].

[6] N. Cardoso and R. Abreu, "MHS2: A Map-Reduce Heuristic-Driven Minimal Hitting Set Search Algorithm," in *Proc. of the International Conference on Multicore Software Engineering, Performance, and Tools (MUSEPAT'13), St. Petersburg, Russia*, ser. Lecture Notes in Computer Science, vol. 8063. Springer Berlin Heidelberg, August 2013, pp. 25–36.

[7] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE Journal Selected Areas in Communications*, vol. 23, no. 10, pp. 2069–2084, October 2005.

[8] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn, "Design and implementation of a TCG-based Integrity Measurement Architecture," in *Proc. of the 13th USENIX Security Symposium (SSYM'04), San Diego, California, USA*. ACM, August 2004, pp. 223–238.

[9] E. Cesena, G. Ramunno, R. Sassu, D. Vernizzi, and A. Lioy, "On scalability of remote attestation," in *Proc. of the 6th ACM workshop on Scalable Trusted Computing (STC'11), Chicago, Illinois, USA*. ACM, October 2011, pp. 25–30.

[10] D. Hoffman and K. Yoo, "Blowtorch: A framework for firewall test automation," in *Proc. of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE'05), Long Beach, California, USA*. IEEE, November 2005, pp. 96–103.

[11] J. Jürjens and G. Wimmel, "Specification-based testing of firewalls," in *Proc. of the 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics (PSI'01), Novosibirsk, Russia*, ser. Lecture Notes in Computer Science, vol. 2244. Springer Berlin Heidelberg, July 2001, pp. 308–316.

[12] A. El-Atawy, K. Ibrahim, H. Hamed, and E. Al-Shaer, "Policy segmentation for intelligent firewall testing," in *Proc. of the 1st IEEE Workshop on Secure Network Protocols (NPSEC'05), Boston, Massachusetts, USA*. IEEE, November 2005, pp. 67–72.

[13] D. Senn, D. Basin, and G. Caronni, "Firewall conformance testing," in *Proc. of the 17th International Conferencen on Testing of Communicating Systems (TestCom'05), Montreal, Canada*, ser. Lecture Notes in Computer Science, vol. 3502. Springer Berlin Heidelberg, May 2005, pp. 226–241.

[14] A. R. Khakpour and A. X. Liu, "Quantifying and querying network reachability," in *Proc. of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS'10), Genova, Italy*. IEEE, June 2010, pp. 817–826.

[15] C. Basile, A. Cappadonia, and A. Lioy, "Network-Level Access Control Policy Analysis and Transformation," *IEEE/ACM Transactions on Networking*, vol. 20, no. 4, pp. 985–998, August 2012.

[16] C. Basile, D. Canavese, A. Lioy, and F. Valenza, "Inter-technology conflict analysis for communication protection policies," in *Proc. of the 9th International Conference Risks and Security of Internet and Systems (CRiSIS14), Trento, Italy*, ser. Lecture Notes in Computer Science, vol. 8924. Springer Berlin Heidelberg, August 2015, pp. 148–163.

[17] C. Basile, D. Canavese, A. Lioy, C. Pitscheider, and F. Valenza, "Inter-function anomaly analysis for correct SDN/NFV deployment," *Networks*, vol. 26, no. 1, pp. 25–43, January 2016.

[18] J. G. Alfaro, N. Boulahia-Cuppens, and F. Cuppens, "Complete analysis of configuration rules to guarantee reliable network security policies," *International Journal of Information Security*, vol. 7, no. 2, pp. 103–122, April 2008.

[19] A.-R. Sadeghi and C. Stüble, "Property-based attestation for computing platforms: caring about properties, not mechanisms," in *Proc. of the 2004 ACM Workshop New Security Paradigms, (NSPW'04), Nova Scotia, Canada*. ACM, September 2004, pp. 67–77.

[20] J. A. Poritz, "Trusted in Computing, Signed Code and the Heat Death of the Internet," in *Proc. of the 2006 ACM Symposium on Applied Computing (SAC'06), New York, New York, USA*. ACM, April 2006, pp. 1855–1859.

[21] X.-Y. Li, C.-X. Shen, and X.-D. Zuo, "An efficient attestation for trustworthiness of computing platform," in *Proc. of the International Conference on Intelligent Information Hiding and Multimedia (IIH-MSP'06), Pasadena, California, USA*. IEEE, December 2006, pp. 625–630.

[22] M. Alam, X. Zhang, M. Nauman, T. Ali, and J.-P. Seifert, "Model-based behavioral attestation," in *Proc. of the 8th ACM Symposium on Access Control Models And Technologies (SACMAT'08), Estes Park, Colorado, USA*. ACM, June 2008, pp. 175–184.

[23] K. Goldman, R. Sailer, D. Pendarakis, and D. Srinivasan, "Scalable integrity monitoring in virtualized environments," in *Proc. of the 5th ACM workshop on Scalable Trusted Computing (STC'10), Chicago, Illinois, USA*. ACM, October 2010, pp. 73–78.

[24] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh, "Terra: a virtual machine-based platform for trusted computing," in *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP'03),*

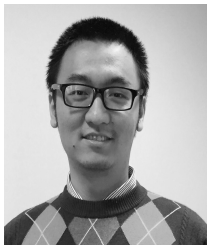
*Bolton Landing, New York, USA*. ACM, October 2003, pp. 193–206.

[25] J. Schiffman, H. Vijayakumar, and T. Jaeger, "Verifying System Integrity by Proxy," in *Proc. of the 5th International Conference on Trust and Trustworthy Computing (TRUST'12), Vienna, Austria*, ser. Lecture Notes in Computer Science, vol. 7344. Springer, Berlin, Heidelberg, June 2012, pp. 179–200.

---

## Author Biography

**Fulvio Valenza** received the M.Sc. degree (summa cum laude) in computer engineering in 2013 from the Politecnico di Torino, where he is currently a third year PhD student. He is working as a Researcher with the Institute of Electronics, Information Engineering, and Telecommunications, National Research Council of Italy, Torino, Italy. His research activity focuses on network security policy, with particular emphasis on policy conflict analysis, resolution, comparison, refinement and optimization.



**Tao Su** received his BS in Telecommunication Engineering and MS in Computers and Communication Network Engineering both from Politecnico di Torino (Italy), where he is currently working toward his PhD degree as a research assistant in the TORSEC research group. His research interests include distributed system security, mobile computing system security, and trusted computing.

**Serena Spinoso** received her M.Sc.Degree (summa cum laude) in Computer Engineering in 2013 from Politecnico di Torino, Turin, Italy. She is currently a Ph.D student in Computer and Control Engineering in the same university. Her research interests include techniques for configuring network functions in NFV-based networks and formal methods applied to verify forwarding correctness of SDN-based networks.

**Antonio Lioy** is full professor at the Politecnico di Torino, where he leads the TORSEC research group active in information system security. His research interests include network security, policy-based system protection, and electronic identity. Lioy received a M.Sc. in Electronic Engineering (summa cum laude) and a Ph.D. in Computer Engineering, both from the Politecnico di Torino.

**Riccardo Sisto** received the M.S. degree in Electronic Engineering in 1987, and the Ph.D. degree in Computer Engineering in 1992, both from Politecnico di Torino, Torino, Italy. Since 1991 he has been working at Politecnico di Torino, in the Computer Engineering Department, first as a Researcher, then as an Associate Professor and, since 2004, as a Full Professor of Computer Engineering. His main research interests are in the area of formal methods, applied to software and communication protocol engineering, distributed systems, and computer security. He has authored and co-authored more than 100 scientific papers. Dr. Sisto is a Senior Member of the ACM.

**Marco Vallini** received his M.Sc. in Computer Engineering and Ph.D. in Information and System Engineering from the Politecnico di Torino. He is a research assistant in TORSEC group at the Department of Control and Computer Engineering of Politecnico di Torino. His research interests include policy-based system protection, allocation and optimization of security applications and critical infrastructure protection.